****

**REPUBLIQUE DU CAMEROON**

**PAIX-Travail-Patrie**

**MINISTRE DE L’ENSEIGNEMENT SUPERIEUR**

**FACULTE D’INGINERIE**

**ET TECHGNOLOGIE**

**REPUBLIC OF CAMEROON**

**Peace-Work-Fatherland**

**MINISTER OF HIGHER EDUCATION**

**FACULTY OF ENGINEERING**

**AND TECHNOLOGY**

**CEF440:INTERNET PROGRAMMING**

**Presented by:**

**- ABANDA SERGIO FE22A131**

**- ABILATEZIE VIN-WILSON ANU FE22A132**

**- AGBOR EMMANUEL NCHENGE FE22A138**

**- ASHLEY TAN WACHE FE22A154**

**- ⁠ASHU DESLEY FE22A155**

**INSTRUCTOR**: **Dr. NKEMENI VALERY**

**March 2025**

Contents

[General introduction 3](#_Toc194353390)

[1. REVIEW AND COMPERISON OF MAJOR TYPES OF MOBILE APPS 3](#_Toc194353391)

[1.1. Native Apps: 3](#_Toc194353392)

[1.2. PROGRESSIVE WEB APPS (PWAs) 4](#_Toc194353393)

[1.3. HYBRID APPS 4](#_Toc194353394)

[2. Mobile App Programming Languages 5](#_Toc194353395)

[A. Native Mobile App Development 5](#_Toc194353396)

[iOS Development 5](#_Toc194353397)

[Android Development 6](#_Toc194353398)

[Decision Factors and Use Cases 8](#_Toc194353399)

[3. Review and compare mobile app development frame works 8](#_Toc194353400)

[OVERVIEW OF LEADING MOBILE APPLICATION FRAMEWORKS 8](#_Toc194353401)

[REACT NATIVE 8](#_Toc194353402)

[FLUTTER 9](#_Toc194353403)

[IONIC 9](#_Toc194353404)

[APPACHE CORDOVA (PhoneGap) 9](#_Toc194353405)

[NATIVE SCRIPT 10](#_Toc194353406)

[APPCELERATOR TITANUM 10](#_Toc194353407)

[4.MOBILE APP ARCHITECTURES AND DESIGN PATTERNS 10](#_Toc194353408)

[Architectures: 10](#_Toc194353409)

[1.Monolithic Architecture: 11](#_Toc194353410)

[2. Microservices Architecture: 11](#_Toc194353411)

[5. Collecting and Analyzing Users Requirements for a Mobile Application 14](#_Toc194353412)

[Introduction 14](#_Toc194353413)

[1. Collecting User Requirements 14](#_Toc194353414)

[2. Documenting User Requirements 15](#_Toc194353415)

[3. Analyzing User Requirements 15](#_Toc194353416)

[6. Estimation of Mobile App Development Costs 16](#_Toc194353417)

[Conclusion 16](#_Toc194353418)

# General introduction

In the rapidly evolving digital landscape, mobile applications have become a cornerstone of modern technology, influencing how users interact with services and information. This report focuses on the fundamental aspects of mobile programming, exploring the various types of mobile applications, the programming languages used in their development, and the methodologies employed to ensure their success.

## REVIEW AND COMPARISON OF MAJOR TYPES OF MOBILE APPS

**INTRODUCTION:**

In the world of mobile application development, three major types of apps dominate the landscape, Native Apps, Progressive Web Apps and Hybrid Apps. Each type has its unique characteristics, advantages, and disadvantages.

## 1.1. Native Apps:

Native apps are developed specifically for a particular operating system (OS) such as IOS or ANDRIOD. They are build using platform specific programming languages ( for example swift for IOS and Kotlin for Android )

**Advantages of native apps**

**Performance:** native apps offer high performance and responsiveness, as there are built for a particular operating system.

**User experience:** There provide a seamless user experience and can utilize device features like camera, GPS and notifications.

**Disadvantages of Native Apps**

**Cost:** Developing a native app for both IOS and Android requires separate codebases and development teams, making it more expensive and time-consuming.

**Maintenance:** Updates and maintenance need to be done separately for each platform, increasing effort and cost.

**Installation:** Users must download and install the app from an app store, which can be a barrier for some.

## 1.2. PROGRESSIVE WEB APPS (PWAs)

PWAs are applications that functions like native apps but are access via a web browser. There can be installed on the device screen without going through the app store. There are build using standard web technologies (HTML, CSS, JavaScript).

**Advantages of PWAs**

**Cross-platform:** PWAs work on any device with a web browser, eliminating the need for several versions.

**Low development cost:** PWAs are cheaper and faster to develop than native apps because there's a single codebase for all platforms.

**No App Store Needed:** Users don’t need to download an app from an app store; they can simply visit the website, and it can be added to their home screen.

**Offline Access:** they can work offline or in low network conditions, thanks to service networks.

**Automatic Updates:** PWAs can be updated automatically, so users always have the latest version without needing to download updates manually.

**Disadvantages of PWAs**

**Limited Device Access:** PWAs have limited access to device features compared to native apps. For example, full access to sensors, Bluetooth, and other native APIs may not be possible.

**Performance:** While PWAs can perform well, they may not offer the same speed and responsiveness as a native app, especially for complex or resource-heavy applications.

**Browser Compatibility:** Some older browsers may not support the full range of PWA capabilities, though this is improving over time.

## 1.3. HYBRID APPS

Hybrid apps combine both elements of native and progressive web app. There are build using web technologies and are wrapped in a native shell, allowing them to be distributed through app store.

**Advantages:**

**Cross-Platform:** Like PWAs, hybrid apps can run on multiple platforms with a single codebase.

**Cost and Time Efficiency:** Hybrid apps are cheaper and faster to develop than native apps because they share code across platforms.

**Access to Device Features:** Hybrid apps can access many native features (e.g., camera, GPS) via plugins or APIs, though they may not have the same level of access as a fully native app.

**Disadvantages:**

**Performance Issues:** Hybrid apps typically do not perform as well as native apps, especially for complex or resource-intensive tasks. They may suffer from slower animations or reduced responsiveness.

**User Experience:** The user experience can sometimes feel less "native" than with fully native apps. There may be inconsistencies in performance and design across platforms.

**Dependence on Frameworks:** The app's performance and features can be limited by the capabilities of the hybrid framework used (e.g., Cordova, Ionic).

# 2. **Mobile App Programming Languages**

Mobile app development primarily relies on several programming languages, each with strengths and weaknesses. The choice depends on factors like **performance, development speed,** and **platform compatibility.**

Below is a comprehensive review and comparison of the key mobile app programming languages, covering **native development, cross-platform frameworks,** and **progressive web apps.**

## Native Mobile App Development

Native development means writing apps specifically for one platform. This approach typically yields high performance and seamless integration with device hardware and OS features.

### iOS Development

* **Swift**
  + **Pros**:
    - Modern, expressive syntax that’s easier to read and maintain
    - High performance and safety features to avoid common bugs
    - Strong integration with Apple’s ecosystem and tools
  + **Cons**:
    - Limited to Apple platforms
    - Smaller community compared to languages used across platforms
* **Objective-C**
  + **Pros**:
    - Mature language with decades of usage in many legacy apps
    - Vast amount of existing libraries and codebases
  + **Cons**:
    - More verbose and complex syntax compared to Swift
    - Slower evolution in terms of language features and safety improvements

### Android Development

* **Kotlin**
  + **Pros**:
    - Officially supported by Google with modern syntax and null-safety features
    - Interoperability with Java, making migration easier
    - Reduces boilerplate code, leading to faster development
  + **Cons**:
    - Although growing fast, the talent pool is still catching up with that of Java
    - Sometimes, compilation times can lag compared to Java
* **Java**
  + **Pros**:
    - Extremely popular with a massive ecosystem, libraries, and community support
    - Long history of stability and performance in enterprise environments
  + **Cons**:
    - Verbose and can lead to boilerplate-heavy code
    - Lacks some of the modern language features and safety that Kotlin offers

1. **Cross-Platform Development**

These frameworks allow developers to write a single codebase for both iOS and android, reducing time and cost. However, they sometimes trade off raw performance or direct access to native APIs.

* **Flutter (Dart)**
* **Pros**:
  + Provides a rich set of customizable widgets for a highly polished UI
  + Delivers near-native performance through its compiled code
  + A single codebase reduces development and maintenance efforts
* **Cons**:
  + Generally results in larger app sizes
  + The ecosystem, while rapidly growing, is still newer compared to native languages
* **React Native (JavaScript)**
* **Pros**:
  + Leverages JavaScript, which many developers already know
  + Offers fast development cycles thanks to hot reloading
  + Has a strong community and a wide range of third-party libraries
* **Cons**:
  + Relies on a bridge to interact with native components, which may introduce performance overhead
  + Complex native integrations may require additional native code
* **Xamarin/MAUI (.NET)**
* **Pros**:
  + Uses C#, allowing developers familiar with the .NET ecosystem to create mobile apps
  + Xamarin has been around for years and is backed by Microsoft
  + MAUI is evolving as a unified framework for building native apps across multiple platforms
* **Cons**:
  + Larger app sizes and sometimes slower performance compared to fully native solutions
  + Fewer community resources and third-party libraries relative to more established platforms like Flutter or React Native

1. **Progressive Web Apps (PWAs)**

PWAs are built with web technologies (HTML, CSS, and JavaScript) and run in the browser but can offer an app-like experience.

* **Pros**:
  + No need for app store approval, which can speed up deployment
  + Easier updates since the code is hosted on the web
  + Works across multiple devices and platforms without platform-specific changes
* **Cons**:
  + Limited access to device hardware and native features
  + Performance can be less optimal compared to native or cross-platform apps
  + Offline capabilities are improving but still not as robust as native apps

### Decision Factors and Use Cases

| **Use Case** | **Recommended Approach** |
| --- | --- |
| **High-Performance, Device-Specific Apps** | Swift for iOS, Kotlin for Android |
| **Rapid Development with a Single Codebase** | Flutter or React Native |
| **Enterprise or .NET Ecosystem Apps** | Xamarin/MAUI |
| **Broad Compatibility and Web-Like Experience** | Progressive Web Apps (PWAs) |

* **Analysis Summary**

By comparing these languages and frameworks, it’s clear that:

* **Native development** (Swift, Objective-C, Kotlin, and Java) remains the best choice for performance-critical applications or when deep integration with device hardware is necessary.
* **Cross-platform frameworks** like **Flutter** and **React Native** offer compelling benefits for businesses aiming to reduce development time and cost, though they might not always match native performance.
* **Progressive Web Apps** are a solid option for applications where ease of deployment and broad reach are more important than native-level performance or access to device features.

# 3. Review and compare mobile app development frame works

INTRODUCTION

The mobile application landscape is rapidly evolving, necessitating the use of frameworks that streamline development processes while ensuring optimal performance and user satisfaction. This report aims to assess the current mobile application frameworks to provide a clear framework for future projects.

## OVERVIEW OF LEADING MOBILE APPLICATION FRAMEWORKS

### REACT NATIVE

**Language:** JavaScript/TypeScript

**Description:** Developed by Facebook, React Native allows for the building of mobile applications using React, combining native components for enhanced performance.

**Key Features:**

* Hot reloading for immediate feedback during development.
* Extensive ecosystem of third-party libraries.
* Strong community support with active contributions.

### FLUTTER

**Language:** Dart

**Description:** A UI toolkit from Google for building natively compiled applications across mobile, web, and desktop platforms from a single codebase.

**Key Features:**

* Rich widget library for customizable and expressive UIs.
* High performance due to direct compilation to native code.
* Strong support for animations and graphics rendering.

### IONIC

**Language:** HTML, CSS, JavaScript

**Description:** A hybrid framework that enables the development of mobile applications using web technologies, built on Angular (or React/View).

**Key Features:**

* Extensive library of pre-built UI components
* Support for Progressive Web Applications (PWAs)
* Easy integration with popular front-end frameworks

### APPACHE CORDOVA (PhoneGap)

**Language:** HTML, CSS, JavaScript

**Description:** A platform for building hybrid mobile applications that wrap web applications in a native container, allowing access to device features.

**Key Features:**

* Broad platform compatibility and support
* Extensive plugin ecosystem for added functionality
* Rapid development cycle utilizing web technologies

XAMARIN

**Language:** C#

**Description:** A Microsoft-owned framework that allows developers to create cross-platform applications with a shared codebase, providing a native user experience.

**Key Features:**

* Access to native APIs for platform-specific functionality
* Strong integration with Visual Studio for development
* Code sharing capabilities across platforms

### NATIVE SCRIPT

**Language:** JavaScript, TypeScript, Angular, Vue

**Description:** A framework that allows developers to build truly native apps with JavaScript and XML for UI components, providing direct access to native APIs.

**Key Features:**

* Native performance and user interfaces
* Direct access to native APIs without additional wrappers
* Strong support for modern web frameworks

### APPCELERATOR TITANUM

**Language:** JavaScript

**Description:** A framework for building native applications using a single codebase written in JavaScript, compiling to native code for enhanced performance.

**Key Features:**

* Rapid application development capabilities
* Access to native device features for enhanced functionality
* Strong enterprise support and scalability

# 4.MOBILE APP ARCHITECTURES AND DESIGN PATTERNS

Mobile Application architectures and Design Patterns define how apps are structured, how components interact, and how maintainability, scalability, and performance are ensured. Common architectures include; Layered, Monolithic, and Microservices, While design patterns like; MVC, MVP, and MVVM help structure code and UI.

## Architectures:

1. **Layered Architecture:**

It Organizes the app into layers (presentation, business logic, data access) for modularity and easier maintenance, especially for large, complex apps.

However, layered architecture also has its downsides. It can lead to high coupling between layers, making changes difficult to implement. Also, changes in one layer can affect all layers above it, leading to potential instability.

## 1.Monolithic Architecture:

In this Architecture, all Components of the software are interconnected and interdependent, and the components cannot function independently of one another. It packages all components into a single unit, suitable for smaller, simpler apps, which makes the application easy to develop, test, and deploy. However they can be harder to scale and maintain, especially as the size and complexity of the application increase.

## Microservices Architecture:

Here, the application is built as a collection of small, independent, deployable services, enhancing scalability and flexibility, and also each service runs in its own process and communicates with others via well-defined APIs.

As others it also presents challenges. It can lead to complex systems and require substantial operational overhead. Furthermore, the need for inter-service communication can result in network latency and data consistency issues.

Design Patterns:

* ***Model-View-Controller (MVC):***

This is a Design Model that separates an application into three interacting parts which are: Model, View, and Controller. This separation allows for better code design and Modularization.

- Model: Represents application data and business logic.

- View: Displays data to the User

- Controller: Processes user input and controls data flow between Model and View

* ***Model-View-Presenter (MVP):***

This is a design model that separates an application into three interacting parts which are: Model, View and Presenter. This is similar to MVC but puts more responsibility on the Teacher to manage the interaction between Model and View.

- Model: Represents application data and business logic.

- View: Displays data to the User

- Presenter: Acts as an intermediary processing user input and updating the View and Model.

* ***Model-View-ViewModel (MVVM):***

MVVM is a design model widely used in mobile development, especially in frameworks like Android’s Jetpack. Its purpose is to separate the application into three parts: Model, View, and ViewModel.

- **Model**: Represents data and business logic.

-**View**: Represents the user interface.

-**ViewModel**: Acts as an interface between the Model and the View, which contains the reference logic.

* ***VIPER (View-Interactor-Presenter-Entity-Router):***

VIPER stands for View, Interactor, Presenter, Entity, and Router. VIPER is primarily based at the clean architecture ideas, which purpose to separate the concerns of different layers of the utility. Each layer has a single duty and communicates with different layers through properly-defined interfaces.

- *View:*

The view is chargeable for showing the information provided by way of the presenter and forwarding the person moves to the presenter.

- *Presen*

The presenter is liable for fetching the records from the interactor, reworking it right into a suitable layout for the view, and updating the view hence. The presenter additionally handles the consumer movements acquired from the view and calls the router to navigate to other screens

- *Interactor:*

The interactor is accountable for gaining access to the facts from the service layer, acting any vital operations on it, and returning it to the presenter. The interactor additionally communicates with the entity layer to store and retrieve the information fashions.

- *Entity*:

The entity is responsible for representing the data in a constant and coherent manner throughout the software. The entity layer also can encompass records get entry to gadgets (DAOs) or repositories that summary the information of records patience and retrieval.

- *Router*:

The router is chargeable for developing and providing the view controllers, passing any vital facts to them, and coping with any dependencies or configurations. The router also communicates with the presenter to get hold of the navigation requests and execute them.

* ***Singleton:***

The singleton policy ensures that there is only one instance of a class and provides global access. This is especially useful when you want to manage a single instance of an object or control access to a delayed object.

For example: Singleton can be used to manage player’s score in mobile game. There can only be one instance that is responsible for tracking scores and is updated throughout the game.

* ***Factory Method:***

The Factory Method model defines an interface for creating an object but allows subclasses to modify the type of the created object. Especially useful when you need to create objects with a common interface but different functionality.

For Example: In a mobile app that supports multiple payment gateways, payments can be made using the Factory Method. Each payment gateway (e.g., PayPal, Stripe) is a small business and provides its services.

* ***Observer:***

The observer structure defines one to many dependencies between objects, so when one object changes its state, all its dependents are automatically notified and updated. This is useful for scheduling distributed events.

For Example: In the reports app, many features (Observers) such as the title widget, the report feed view, and the notification provider (Themes) can subscribe to updates when new information arrives. The observer model ensures that they are created all registered parts report, and accordingly You can update it.

* ***Dependency Injection (DI):***

Dependency Injection is a method of providing class dependencies from the outside, rather than creating them in the class. It improves code modularity and testability by making classes independent of their dependencies.

For Example: In an Android app, instead of creating a single database connection object in the class, you can place the database object externally, allowing you to easily test and modify database operations.

# 5. Collecting and Analyzing Users Requirements for a Mobile Application

## Introduction

Requirement engineering is a crucial process in software development that ensures a mobile application meets the needs and expectations of its users. It involves collecting, documenting, and analyzing user requirements to create a well-defined set of functional and non-functional specifications. This document outlines the key steps involved in collecting and analyzing user requirements for a mobile application.

## 1. Collecting User Requirements

Collecting user requirements involves gathering information from stakeholders, including end users, clients, and business owners. Several techniques can be used:

1.1 Interviews

- Conduct structured or unstructured interviews with potential users to understand their needs.  
- Ask open-ended questions to gather insights on user expectations and pain points.

1.2 Surveys and Questionnaires

- Distribute online or paper-based surveys to collect responses from a broad user base.  
- Use Likert scales, multiple-choice questions, and open-ended questions to gain both qualitative and quantitative data.

1.3 Focus Groups

- Gather a small group of target users to discuss their needs and preferences.  
- Encourage users to share their opinions and experiences regarding similar applications.

1.4 Observations

- Observe how users interact with similar mobile applications.  
- Identify usability challenges and common usage patterns.

1.5 Competitive Analysis

- Study competitors’ applications to understand existing features, strengths, and weaknesses.  
- Identify gaps that can be addressed in the new mobile application.

## 2. Documenting User Requirements

After gathering the requirements, they must be documented in a structured format:

2.1 Functional Requirements

- Define the specific features and functionalities the application must have.  
- Example: "The app should allow users to register using their email or social media accounts."

2.2 Non-Functional Requirements

- Specify performance, security, and usability expectations.  
- Example: "The app should load within three seconds under normal network conditions."

2.3 Use Cases and User Stories

- Use Cases: Define how users will interact with the system.  
- User Stories: Short descriptions of user goals in the following format:  
   
 Example: "As a user, I want to reset my password so that I can regain access to my account."

2.4 Wireframes and Prototypes

- Create sketches or digital wireframes to visualize the application's layout and navigation.  
- Use tools like Figma, Adobe XD, or Balsamiq for digital prototypes.

## 3. Analyzing User Requirements

Once requirements are collected and documented, they must be analyzed to ensure feasibility and completeness.

3.1 Prioritization

- Categorize requirements based on their importance using frameworks like MoSCoW (Must-have, Should-have, Could-have, Won't-have).

3.2 Feasibility Analysis

- Assess technical, financial, and operational feasibility to determine whether the requirements can be implemented.

3.3 Conflict Resolution

- Identify conflicting requirements from different stakeholders and resolve them through discussions and trade-offs.

3.4 Validation and Verification

- Conduct review sessions with stakeholders to validate requirements.  
- Ensure that documented requirements accurately reflect user needs.

# 6. Estimation of Mobile App Development Costs

Estimating the costs of mobile app development involves several key steps:

1. Define Project Scope:

- Identify the features and functionalities required.

- Outline the platforms (iOS, Android, web) for which the app will be developed.

2. Resource Allocation:

- Determine the size and expertise of the development team, including developers, designers, and project managers.

- Consider additional resources such as QA testers and marketing specialists.

3. Time Estimates:

- Break down the project into phases: design, development, testing, and deployment.

- Estimate the time required for each phase based on complexity and team experience.

4. Cost Factors:

- Hourly Rates: Assess the hourly rates of the development team, which can vary based on location and expertise.

- Technology Stack: Consider costs associated with tools, software licenses, and backend services.

- Maintenance and Updates: Include ongoing costs for updates, bug fixes, and support after the app is launched.

5. Contingency Budget

- Allocate a percentage (typically 10-20%) of the total estimated cost for unforeseen expenses

# Conclusion

In summary, this report highlights key aspects of mobile programming and app development. We examined the different types of mobile applications,native, web, and hybrid ,and the programming languages best suited for each.

We also discussed important design patterns and architectures that help in building effective apps. Gathering user requirements was emphasized as a crucial step for creating applications that truly meet user needs.

accurately estimating development costs is essential for successful project management. This report provides a clear overview of these elements, helping stakeholders understand the main factors that contribute to successful mobile app development.